CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
COORDENAÇÃO DO CURSO TÉCNICO EM ELETRÔNICA

Gabriel Lucas Teles Vilaça

LAUNCHPAD F28027

Volume 2: Clock, SPI, Filtros, RAM/FLASH e Modularização

Belo Horizonte - MG
2018

Gabriel Lucas Teles Vilaça

LAUNCHPAD F28027

Volume 2: Clock, SPI, Filtros, RAM/FLASH e Modularização

Belo Horizonte - MG
2018

Resumo

Apostila feita por Gabriel Lucas Teles Vilaça. Esta apostila em conjunto com a feita por Gustavo Zappulla para introduzir certas funções doTMS320F28027, produzido pela Texas Instruments, e que pertence à família Piccolo da linha C2000 para controle em tempo real.

Esta apostila explica pontos mais avançados do F28027 como: SPI, Filtros, oscilador externo e etc… Enquanto que a do Gustavo explica como fazer a configuração inicial do PWM e do ADC.

A apostila seguiu a metodologia de explicar o funcionamento do módulo e depois colocar um programa exemplo para mostrar como é feita a configuração do mesmo pelo software

Lista de Figuras

Figura 1.1 Gráficos sobre inconsistência e imprecisão da frequência

Figura 1.2 Opções de clock

Figura 2.1 Diagrama em blocos do SPI

Figura 2.2 Ligação do SPI Mestre/Escravo

Figura 2.3 Registrador SPIBRR

Figura 2.4 Registrador LOSPCP

Figura 3.1 Filtro Biquad

Figura 3.2 Número de ciclos de *clock* gastos

Figura 3.3 Opções de outras pastas ao procurar o filtro

Figura 3.4 Gráficos gerados pelo MATLAB

Figura 3.5 Arquivo dos coeficientes gerados

Figura 3.6 Arquivo IIR.h

Figura 4.1 Opções de *Build*

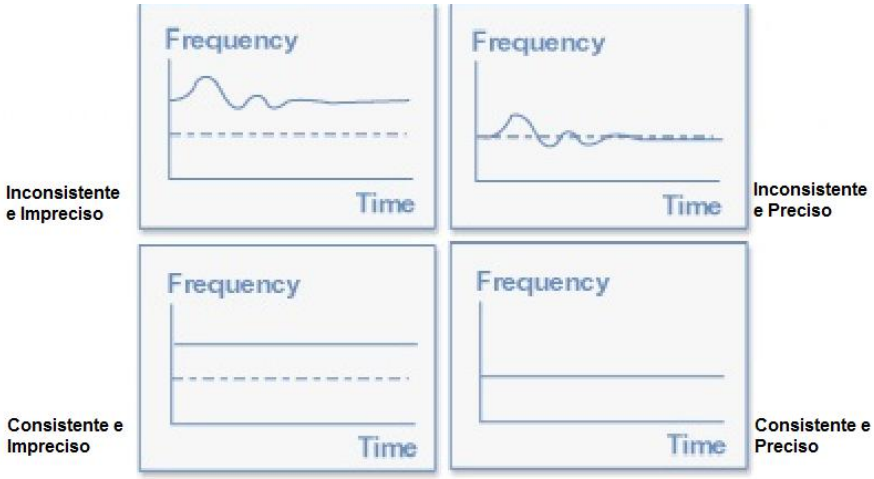
Sumário

1 Clock Externo. 5
1.1 Opções de Clock. 6
1.2 Programa 1. 7
2 Serial Peripheral Interface. 8
2.1 Diagrama em blocos do SPI. 8
2.2 Modos de operação. 9
2.3 LSPCLK e Baud Rate. 10
2.3.1 Erro na função que gera o LSPCLK. 11
2.4 Programa 2. 12
3 Filtros. 13
3.1 Cálculo dos coeficientes do filtro. 15
3.2 Inserindo a biblioteca IIR.h. 16
3.3 Passando os coeficientes ao IIR.h. 17
3.4 Programa 3. 19
4 Memória RAM e FLASH. 20
4.1 Definindo o programa na RAM. 20
4.2 Programa 4. 21
5 Modularização. 22
5.1 Separando as funções. 22
5.2 Programa 5. 23
Anexo 1. 24
Anexo 2. 26
Anexo 3. 31
Anexo 4. 35

1 Clock Externo

O *Clock* gerado pelo oscilador interno está sujeito a problemas de inconsistência e imprecisão que são exemplificados pelos gráficos da figura 1.1 em que a tracejada representa o valor desejado do *clock* do sistema

Figura 1.1: Gráficos sobre inconsistência e imprecisão da frequência



Fonte: http://www.ni.com/product-documentation/4801/en/, acessado em 31/02/2018

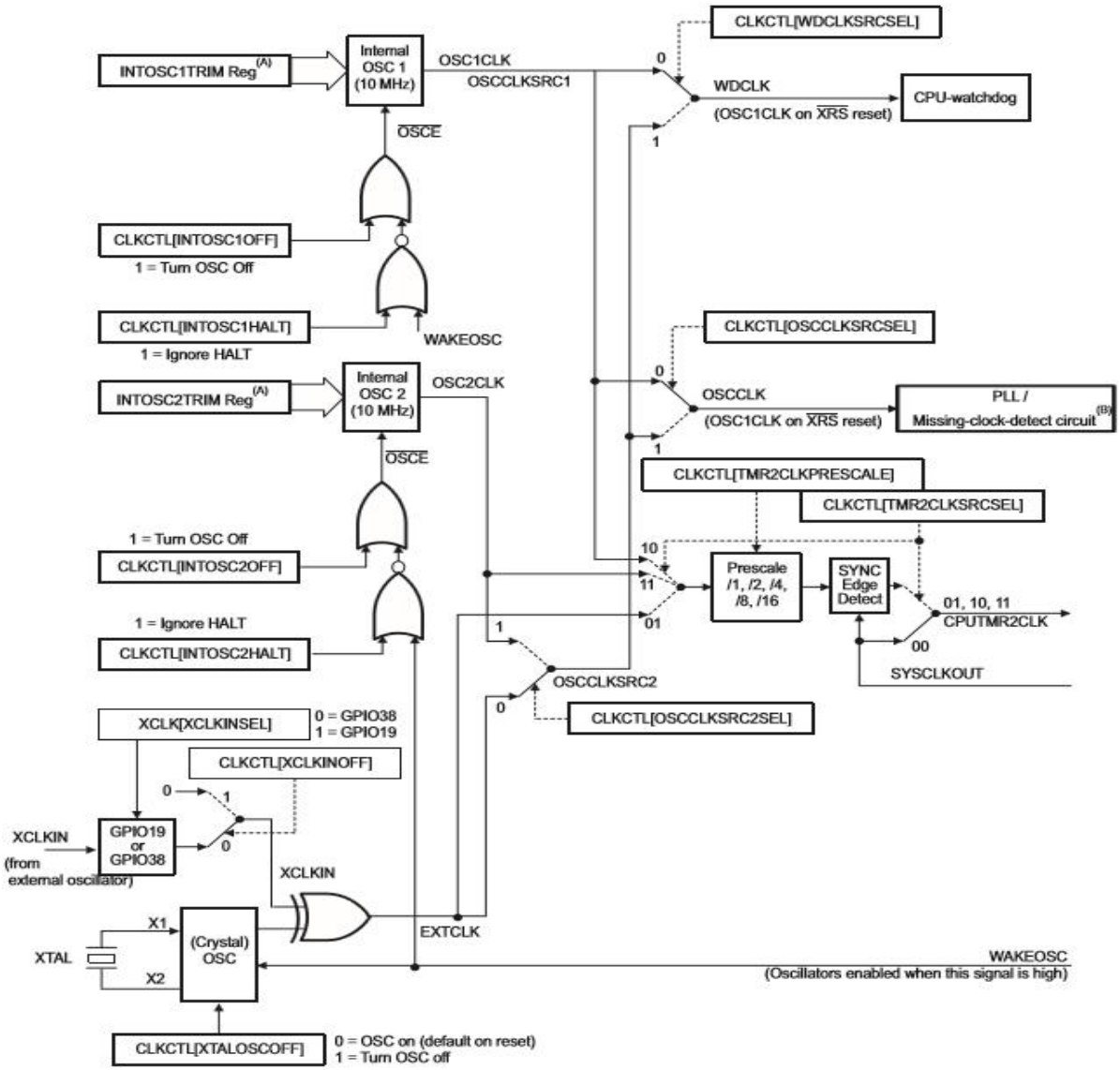
A imprecisão é a frequência media do *clock* é discordante do valor desejado. A inconsistência trata da frequência do *clock* variar em torno da frequência media, sem alterar o valor da frequência media.

Agora, analisando ambos os problemas são graves para controles de malhas fechadas, caso o sistema esteja utilizando algum filtro pois os parâmetros dependem da frequência de amostragem. Então se o *clock* é instável a frequência de amostragem da variável de controle muda e quando esses dados são inseridos no filtro podem gerar ações de controle que podem tirar o sistema de um estado que já está equilíbrio pelo fato de interpretar essas novas medições como erro.

1.1 Opções de Clock

A figura 1.2 mostra as opções de *clock* para o TMS320F28027.

Figura 1.2: Opções de *clock*



Fonte: SPRUFN3D - [5]

Para utilizarmos o oscilador de cristal soldado na placa como fonte de *clock* precisamos configurar as chaves da figura 1.2 de forma a fazer com que o oscilador de cristal seja a nova fonte de *clock*

1.2 Programa 1

O programa 1, disponível no Anexo 1, mostra a seleção do cristal soldado na placa como fonte principal de *clock.* O programa também configura o GPIO18 no modo XCLKOUT para possibilitar a visualização do *clock* em um osciloscópio.

2 Serial Peripheral Interface

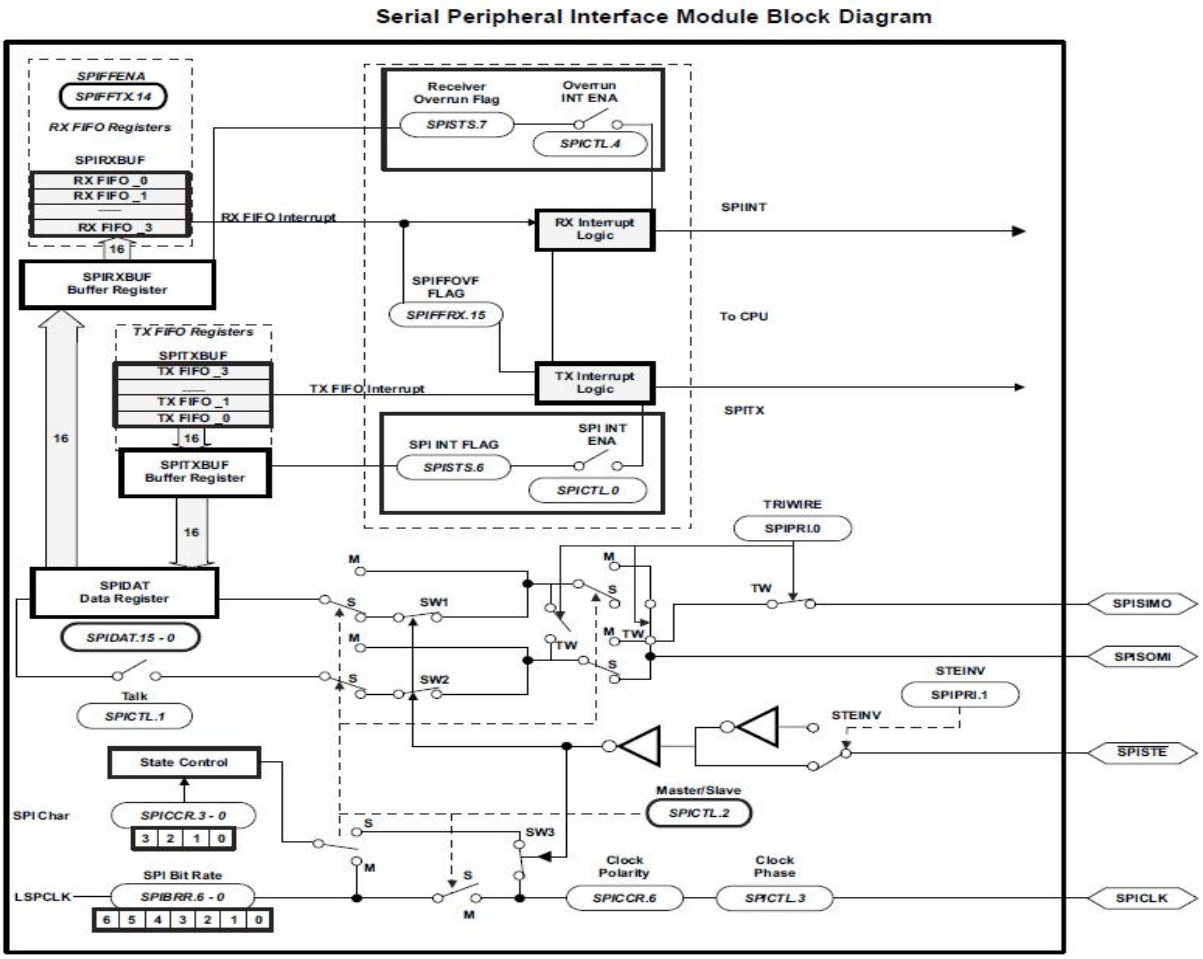
A *Serial Peripheral Interface* (SPI) é um protocolo de interface periférica que transmite informações de forma serial.

É utilizado para fazer dispositivos diferentes se comunicarem. Sendo que existe um mestre e os demais equipamentos conectados na rede se tornam escravos, recebendo ou enviando informações com base no *clock* que é fornecido pelo mestre.

2.1 Diagrama em blocos do SPI

A figura 2.1 mostra o diagrama em blocos do módulo SPI.

Figura 2.1: Diagrama em blocos do SPI



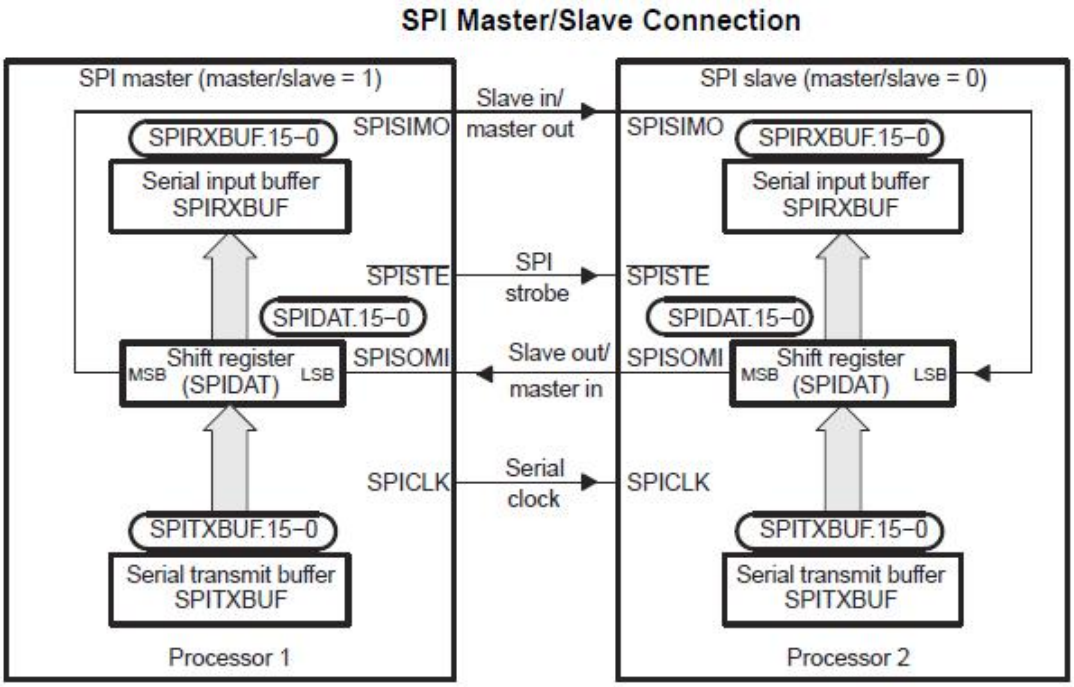
Fonte: SPRUG71B - [11]

2.2 Modos de operação

A comunicação utilizando o modulo SPI dá a possibilidade de escolher entre dois modos mestre e escravo. Apenas o modo mestre será exemplificado, caso o leitor deseje obter mais conhecimento sobre como utilizar o modo escravo mais informações podem ser obtidas no SPRUG71B – [11]

A figura 2.2 mostra como é a ligação do SPI entre o mestre e o escravo. O SPICLK é o *clock* que determina a velocidade que as informações são enviadas ou recebidas. SPISIMO é a saída de dados do mestre e a entrada de dados no escravo. SPISOMI é a saída de informação do escravo e entrada delas no mestre. SPISTE é responsável por habilitar e desabilitar a transmissão do escravo e é ativo em baixo. Um detalhe importante a ressaltar é que o dispositivo mestre é responsável por gerar o SPICLK.

Figura 2.2 Ligação do SPI Mestre/Escravo



Fonte: SPRUG71B – [11]

2.3 LSPCLK e Baud Rate

Low Speed Peripheral Clock (LSPCLK) é o *clock* de baixa velocidade. É utilizado nos módulos de comunicação disponíveis no módulo que são: I2C, SPI e SCI. Seu valor é a divisão do *clock* principal do sistema por: 1, 2, 4, 6, 8, 10, 12 ou 14. O valor do divisor é escolhido pelo programador e por padrão o valor do divisor é 4 após um *reset*.

Baud Rate é a taxa de transmissão ou recepção de dados do sistema. Para alterar o valor do *Baud Rate* devemos alterar o valor do registrador SPI *Baud Rate Register*(SPIBRR), que pode ser visto na figura 2.3

Figura 2.3 Registrador SPIBRR

SPI Baud Rate Register (SPIBRR) — Address 7044h

7	6	5	4	3	2	1	0
Reserved	SPI BIT RATE 6	SPI BIT RATE 5	SPI BIT RATE 4	SPI BIT RATE 3	SPI BIT RATE 2	SPI BIT RATE 1	SPI BIT RATE 0
R-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Fonte: SPRUG71B – [11]

Os bits 0-6 de SPIBRR podem gerar um valor entre 0 até 127, esse valor será usado no calculo do *Baud Rate*. No que se refere ao bit 7 escritas nele não fazem efeito.

O valor do Baud Rate do SPI pode ser calculado pelas formulas:

SPIBRR tem seu valor igual 0, 1 ou 2:

$$\text{SPI Baud Rate} = \frac{LSPCLK}{4}$$

SPIBRR tem um valor de 3 até 127:

$$\text{SPI Baud Rate} = \frac{LSPCLK}{(SPIBRR+1)}$$

O valor de SPI *Baud Rate* é o valor do SPICLK.

2.3.1 Erro na função que gera o LSPCLK

Ao programar o divisor do LSPCLK foi constatado que existe um erro na função CLK_setLowSpdPreScaler(CLK_Handle clkHandle, const CLK_LowSpdPreScaler_e preScaler) que faz uma atribuição errada no registrador LOSPCP, que pode ser visto na figura 8.4. Este registrador é responsável por atribuir o valor do divisor do clock para criar o LSPCLK isso ocasionava problemas ao tentar utilizar o SPI em sua frequência máxima de 15MHz. Foi criada uma função chamada CLK_setLowSpdPreScaler2 para atribuir de forma correta o valor do divisor de LOSPCP, os parâmetros da função criada são os mesmos da função original.

Figura 2.4: Registrador LOSPCP

Low-Speed Peripheral Clock Prescaler Register (LOSPCP)

15 … 3	2 … 0
Reserved	LSPCLK
R-0	R/W-010

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Fonte: SPRUFN3D - [5]

Os bits 0-2 do registrador serão utilizados para alterar o valor do divisor do LSPCLK. Quanto aos bits 3-15 são reservados e qualquer escrita neles deve sempre ser 0, esse cuidado foi tomado ao criar a função CLK_setLowSpdPreScaler2. Segue a função abaixo:

```
void CLK_setLowSpdPreScaler2(CLK_Handle clkHandle, const
CLK_LowSpdPreScaler_e preScaler)
{
    CLK_Obj *clk = (CLK_Obj *)clkHandle;
    ENABLE_PROTECTED_REGISTER_WRITE_MODE;
//Habilita escrita em registro protegido
    clk->LOSPCP &= 0XFFF8;
// Somente os bits referentes ao LSPCLK foram zerados
```

```
    clk->LOSPCP |= preScaler;
//Somente os bits referentes ao LSPCLK de LOSPCP foram alterados
    DISABLE_PROTECTED_REGISTER_WRITE_MODE;
//Desabilita a escrita em registro protegido
    return;
}
```

2.4 Programa 2

O programa 2, disponível no Anexo 2, foi feito com o SPI em modo loopback que é um modo criado para teste do módulo SPI em que as ligações de SPISIMO e SPISOMI são feitas internamente de forma que o SPI transmite e recebe a informação.

3 Filtros

Filtros, usados no processamento de sinas, são ferramentas que tem como objetivo de atenuar ou amplificar certas frequências encontradas em um sinal para melhorar a análise do mesmo.

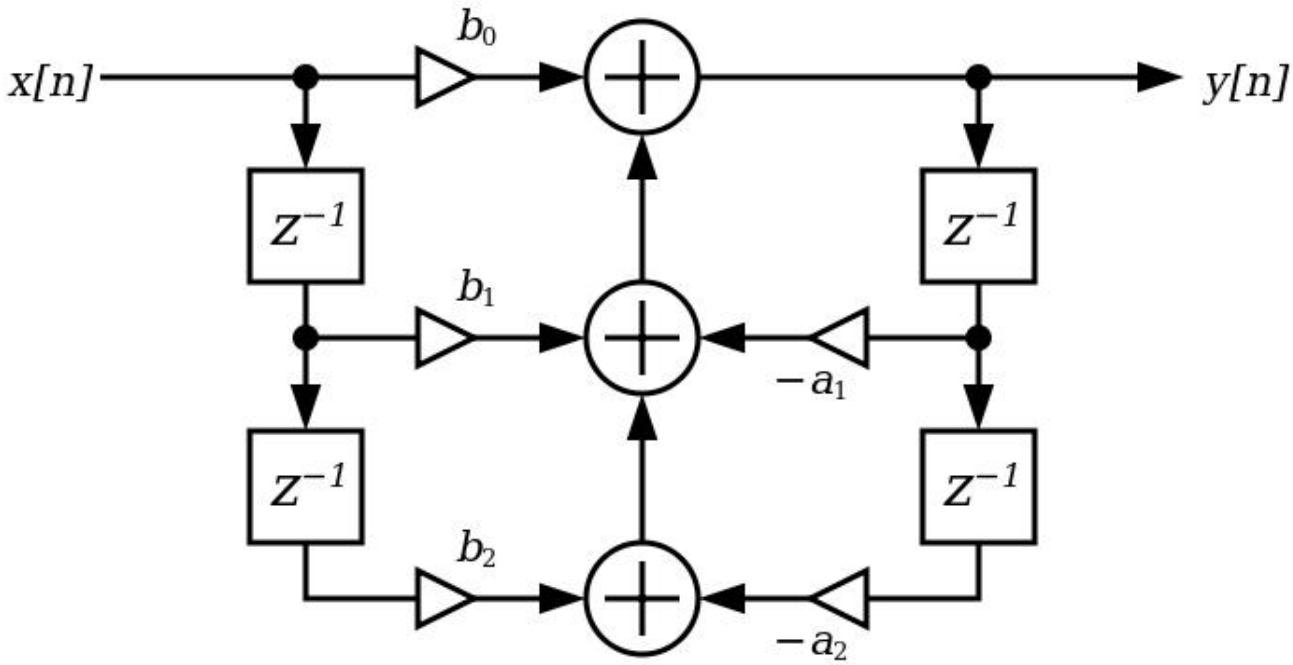
Nessa apostila será abordado o processo de inclusão e utilização da biblioteca de filtros do tipo Infinite Impulse Response (IIR) de 32 bits e seus exemplos serão aplicados ao LaunchXL-F28027. Todos os filtros trabalhados aqui são filtros que usam tempo discreto.

Um filtro IIR simples, constituído de apenas um biquad, que pode ser visto na Figura 1.1, possui uma equação como:

$$y[n]=b_0 x[n]+b_1 x[n-1]+b_2 x[n-2]-a_1 y[n-1]-a_2 y[n-2] \qquad (1)$$

Onde y[n] é o valor da saída atual e x[n] é o valor da entrada atual.

Figura3.1: Filtro Biquad



Fonte: https://en.wikipedia.org/wiki/Digital_biquad_filter

A equação (1) é grande e muito complexa para ser resolvida a mão e mesmo quando feita por um processador pode gerar um grande atraso no programa devido à quantidade de operações envolvidas. É preciso buscar na memória do programa as três últimas entradas e as duas últimas saídas e

multiplicar cada valor por uma constante e então somar todos os resultados para calcular o valor da saída atual.

A equação (1) pode ser generalizada para um filtro IIR genérico de m biquads se tornando:

$$y[n]=\sum_{k=0}^{2m} b_k x_{n-k} - \sum_{k=1}^{2m} a_k y_{n-k} \quad (2)$$

$$y[n]=b_0 x[n]+b_1 x[n-1]+\ldots+b_{2m} x[n-2m]$$

$$-a_1 y[n-1]-a_2 y[n-2]-\ldots-a_{2m} y[n-2m]$$

Felizmente, a Texas Intruments (TI) criou uma biblioteca que é foi optimizada para o calculo de filtros [13] utilizando um buffer, na documentação recebe o nome de *Coefficient Buffer*, que contem os valores dos coeficientes: b_0, b_1, b_2, ..., b_m, a_1, a_2, a_3, ..., a_m .E outro, na documentação recebe o nome de *Delay Buffer*, com os valores de entradas e saídas: x_n,x_{n-1},x_{n-2}, ..., x_{n-m}, y_{n-1}, y_{n-2}, y_{n-3}, ..., y_{n-m}. A TI fez uma pequena tabela com a quantidade de ciclos de clock, do processador, gastos para executar um filtro IIR de 32 bits em função do numero de *biquads*, que pode ser visto na Figura 1.2

Figura3.2: Número de ciclos de *clock* gastos em função da quantidade de *biquads* para um filtro IIR 32 bits

NBIQ	C-Callable ASM
1	22 cycles
2	48 cycles
4	110 cycles
8	202 cycles

Fonte: Fixed Point DSP Software Library – [13]

Analisando os dados da figura 1.2 podemos descobrir quanto tempo será gasto para calcular a saída do filtro da figura 1.1, considerando que estamos operando no *clock* máximo de 60 MHz

$$Tempo=\frac{22*1}{60*10^6} \cong 366{,}67\ ns$$

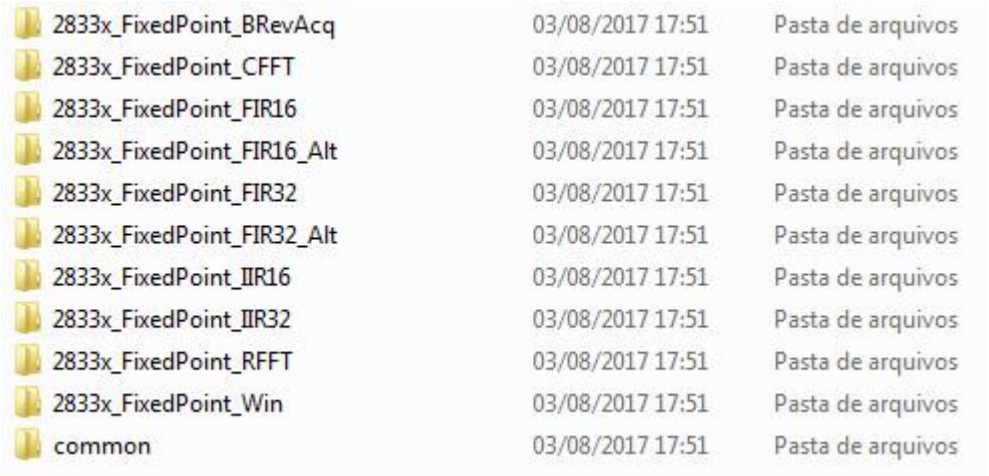
Essa com essa velocidade de calculo é viável a utilização dos filtros no programa.

3.1 Cálculo dos coeficientes do filtro

Para que a biblioteca desenvolvida pela TI seja utilizada devemos inserir os valores dos coeficientes do filtro b_0, b_1, b_2, ..., b_m, a_1, a_2, a_3, ..., a_m para encontrar os valores dos mesmos a TI disponibilizou um programa, feito no MATLAB que devolve um arquivo em formato txt para substituir dentro da biblioteca.

Para acessar o arquivo do MATLAB disponibilizado pela TI é necessário encontrar a pasta onde foi instalado o *Control Suite* (CS) e então seguir: *controlSUITE\libs\dsp\FixedPointLib\v1_20_00_00\examples_ccsv5* . Nessa pasta encontraremos outras opções para calculo de coeficientes de outros tipos de filtro, como pode ser visto na figura 1.3

Figura3.3: Opções de outras pastas ao procurar o programa para calculo dos coeficientes do filtro IIR 32



2833x_FixedPoint_BRevAcq	03/08/2017 17:51	Pasta de arquivos
2833x_FixedPoint_CFFT	03/08/2017 17:51	Pasta de arquivos
2833x_FixedPoint_FIR16	03/08/2017 17:51	Pasta de arquivos
2833x_FixedPoint_FIR16_Alt	03/08/2017 17:51	Pasta de arquivos
2833x_FixedPoint_FIR32	03/08/2017 17:51	Pasta de arquivos
2833x_FixedPoint_FIR32_Alt	03/08/2017 17:51	Pasta de arquivos
2833x_FixedPoint_IIR16	03/08/2017 17:51	Pasta de arquivos
2833x_FixedPoint_IIR32	03/08/2017 17:51	Pasta de arquivos
2833x_FixedPoint_RFFT	03/08/2017 17:51	Pasta de arquivos
2833x_FixedPoint_Win	03/08/2017 17:51	Pasta de arquivos
common	03/08/2017 17:51	Pasta de arquivos

Fonte: Acervo próprio

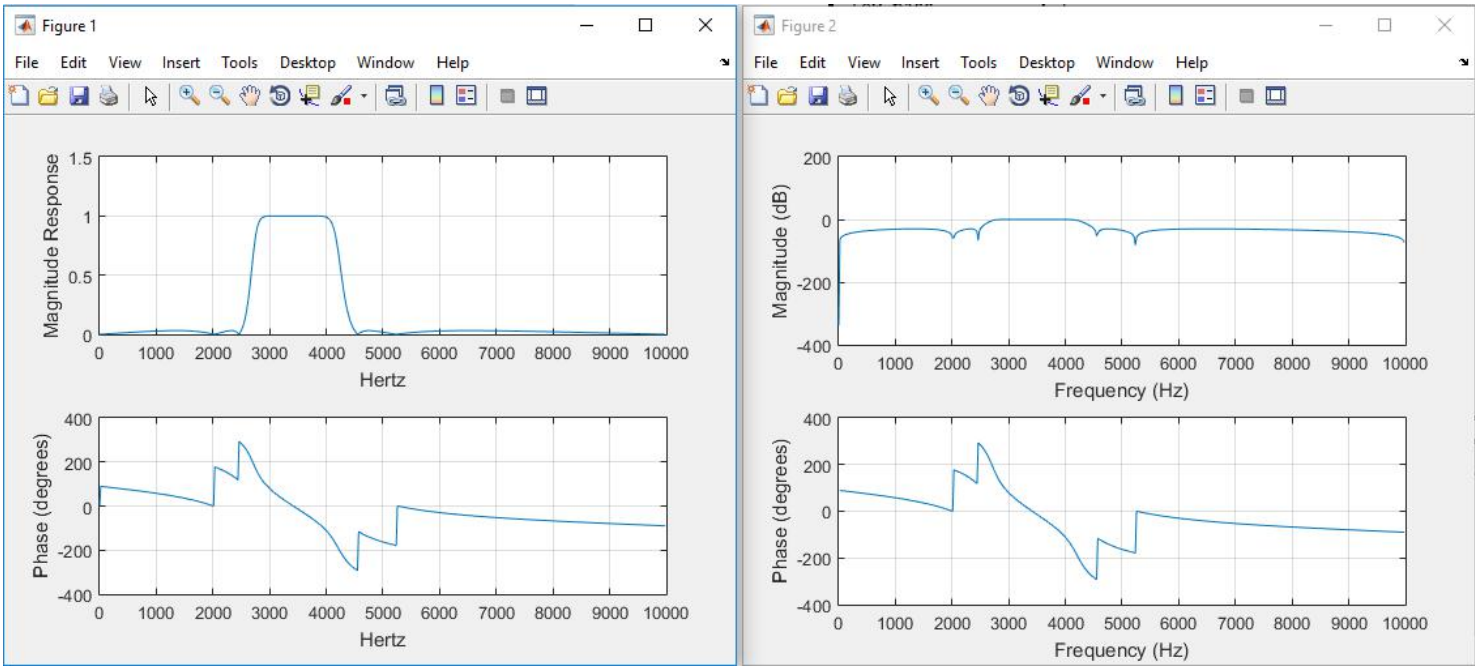
Como o intuito desse tutorial são os filtros IIR de 32 bits escolheremos a pasta 2833x_FixedPoint_IIR32 e em seguida abriremos a pasta matlab. Assim o caminho completo até esse ponto é: *controlSUITE\libs\dsp\FixedPointLib\v1_20_00_00\examples_ccsv5\2833x_FixedPoint_IIR32\matlab.*

Existem duas opções dentro da pasta matlab: eziir32.m e FixedPointIIR32.m. Para calcular os coeficientes abriremos o arquivo eziir32.m.

Ao executar o programa algumas perguntas serão feitas as seguintes perguntas, que com base nas respostas o programa calcula os valores dos coeficientes, sobre o filtro: O tipo de filtro IIR, o tipo de resposta do filtro, frequência de amostragem, ondulação da banda passante em dB, ondulação da banda de corte em dB e por fim o nome do arquivo onde serão guardados os coeficientes.

Apos a execução serão geradas duas janelas pelo MATLAB a primeira janela possui dois gráficos um mostra a magnitude da resposta em função da frequência e o segundo a fase do sinal de saída em função da frequência. A segunda janela possui dois gráficos um mostra a magnitude em dB da resposta em função da frequência e o segundo a fase do sinal de saída em função da frequência. A figura 1.4 mostra um exemplo dessas janelas

Figura 3.4: Gráficos gerados pelo MATLAB



Fonte: Acervo próprio

Segundo os gráficos gerados ao utilizarmos um sinal de 3416Hz com uma frequência de amostragem de 20000Hz a amplitude de entrada e saída devem ser as mesmas a e a fase também.

3.2 Inserindo a biblioteca IIR.H

A primeira coisa que precisamos fazer para possibilitar a utilização do filtro é incluir a biblioter iir.h ao programa. Ao instalar o *ControlSuite* ela pode ser encontrada no seguinte endereço: *controlSUITE\libs\dsp\FixedPointLib\v1_20_00_00\include.* Faça uma copiar de IIR.h e coloque a cópia feita na mesma pasta onde o programa principal está. Dessa forma apenas ao inserir #include "iir.h" no programa o arquivo será encontrado

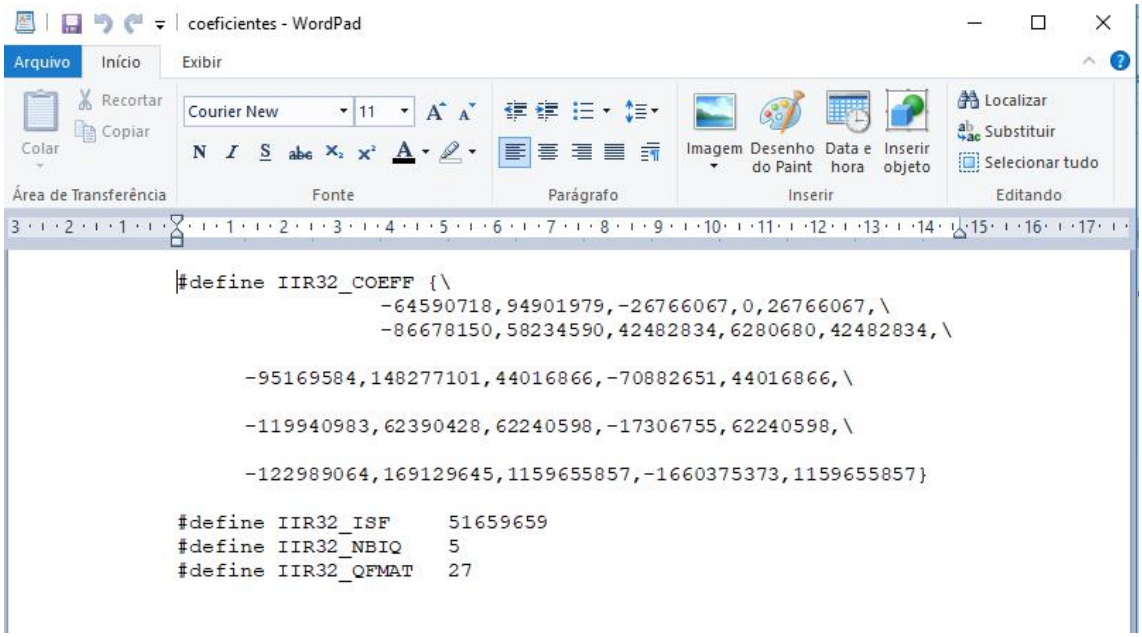
Para que iir.h possa ser utilizado os seguintes passos devem ser executados:

1. Para que o include de iir.h seja efetivado é necessário clicar com o botão direito sobre o projeto e selecionar PROPERTIES.
2. Uma nova janela vai se abrir e no canto superior esquerdo dessa janela escrever FILE SEARCH PATH então no menu "Include library file or command file as input (--library, -l)".
3. Clicar no Add(O documento com o simbolo que ao lado tem o simbolo de adição matemática), uma nova janela vai então se abrir.
4. Nessa janela devemos inserir o caminho da library para o compilador fazer a associação com a do seu projeto então basta adicionar "C:\ti\controlSUITE\libs\dsp\FixedPointLib\v1_20_00_00\lib\c28x_fixedpoint_dsp_library.lib".
5. Uma observação: Esse caminho é valido apenas se a sua pasta TI estiver localizada no diretorio C:, mas caso não esteja basta apenas encontrar a pasta controlSUITE e substituir o que está antes de \controlSUITE

3.3 Passando os coeficientes ao IIR.H

Apos a execução do programa do MATLAB será gerado um arquivo, que o nome é escolhido pelo usuário, ao abrir esse arquivo teremos um texto similar ao da Figura3.5

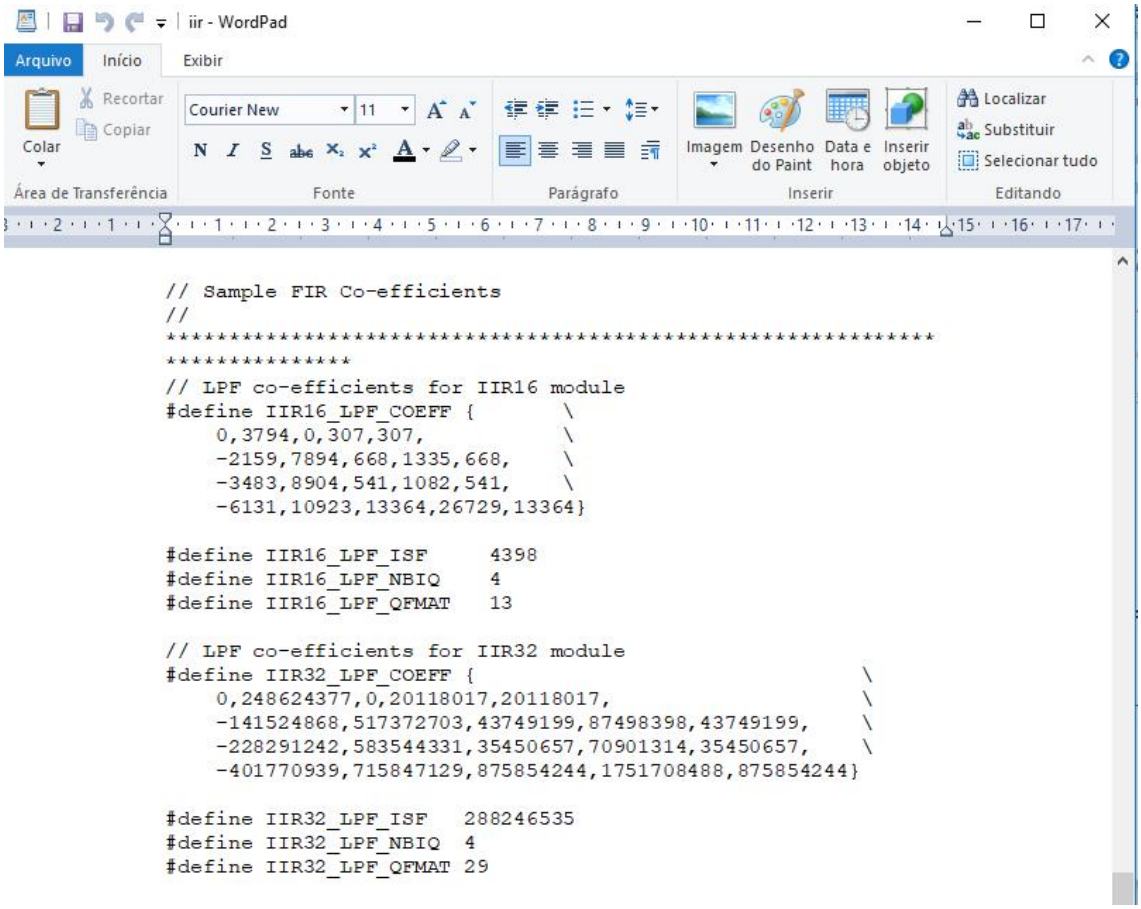
Figura3.5: Arquivo dos coeficientes gerados



Fonte:Acervo próprio

Agora ao abrir a copiar do arquivo IIR.h que foi feita anteriormente, no final do documento, podemos ver um texto similar ao da Figura3.6

Figura3.6: Arquivo IIR.h



Fonte: Acervo Próprio

Agora vamos substituir os valores que já estão presentes no arquivo IIR.h pelos valores que estão no arquivo gerado pelo MATLAB. É muito importante não alterar os valores no IIR.h original pois qualquer erro ao editar pode causar com que a biblioteca funcione de forma errada por isso faremos as alterações na cópia assim o original fica como um *backup*

3.4 Programa 3

O programa, disponível no Anexo 3, mostra os passos para inicializar o filtro e então possibilitar sua utilização em um programa ao gerar um sinal de 60Hz e depois aplicá-lo a um filtro

4 Memória RAM e FLASH

Ao escrever um programa cada vez mais sofisticado é comum que a memória disponível não seja suficiente para comportar todo programa na RAM e como solução a memória de instruções deve ser guardada na *flash* que gasta mais tempo para ser pegue na memória e lida, mas existem partes do código, como as interrupções, que desejamos que possuíssem menor tempo de execução possível.

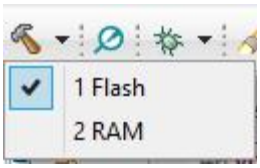
Para isso podemos, usando linhas de comando, mandar o compilador salvar parte do programa na memória RAM e o restante na *flash* que parte do programa fique na memória RAM que é mais veloz que a *flash*, porém possui menos RAM disponível do que *flash* para isso é melhor colocar apenas as funções criticas na memória RAM

IMPORTANTE: Mesmo que você use os comandos para que o compilador faça com que parte, ou todo, o seu programa fique na memória RAM ele sempre vai gravar na *flash* e transferir para a RAM, como medida de segurança. Isso é uma medida de prevenção caso a execução seja interrompida devido a um *reset* ou falta de alimentação ao retornar o conteúdo da RAM, que é volátil, é perdido e o programa precisa recomeçar e caso um backup do programa não tenha sido feito na memória *flash*, que não é volátil, seria preciso esperar até que um novo programa fosse passado para a memória.

4.1 Definindo o programa na RAM

É possível alterar o programa para ser gravado totalmente na RAM ou na FLASH utilizando a sete ao lado do martelo (*build*) como pode ser visto na Figura4.1

Figura 4.1: Opções de *Build*



Fonte: Acervo próprio

Após isso será salvo e executado dentro da memória FLASH e para colocar as funções que se deseja na RAM devemos declarar as funções fora

da main e então depois delas usaremos um #pragma para dizer ao compilador quais funções participam da RAM como pode ser visto no exemplo abaixo:

```
/  Declaração dos protótipos de funções encontradas no
arquivo void funcao_1(void);
int funcao_2(int x); int
funcao_3(void); float
funcao_4(float y);
#pragma CODE_SECTION(funcao_2, "ramfuncs");
#pragma CODE_SECTION(funcao_4, "ramfuncs");
```

Dessa forma as funções função_2 e função_4 serão copiadas para a memória RAM ao inicializar a memória

4.2 Programa 4

O programa, disponível no Anexo 4, apenas faz inicialização básica e então dentro do loop infinito da função main e executa uma das funções DelayRAM e DelayFLASH, onde a primeira está salva na memória RAM e a segunda na memória FLASH.

5 Modularização

Agora que nosso programa está bem dividido na memória, devemos dividir o código em partes, para facilitar o entendimento. Nessa apostila foi optado por dividir o programa em três partes que contem: configurações dos módulos utilizados, rotinas de interrupção e tarefas de baixa prioridade que ficam no loop infinito da função main

A ideia de modularizar é:

- Agrupar partes de configuração do mesmo módulo juntas.
- Simplificar o programa com funções de nomes sugestivos como: Configura_PWM(PWM_Handle), Configura_ADC(), Habilita_Interrupções()
- Comentar qual o objetivo da função, suas entradas e suas saídas.
- Comentar o que uma instrução ou grupo de instruções em conjunto faz.

5.1 Separando as funções

Primeiramente, separamos todas as funções em três arquivos e usamos os seguintes critérios:

- Interrupções.c : Abriga a inicialização da memória e configuração das interrupções
- Modulos_Inicializacao.c : Abriga a configuração dos módulos como: PWM, ADC, SPI, GPIO e etc…
- Principal.c : Contem a função main e um loop infinito contendo tarefas de baixa prioridade

Para fazer a junção entre os três arquivos criamos o arquivo globais.h que contem todos os includes e variáveis que as funções precisam assim os três arquivos compartilham as mesmas variáveis, desde que globais.h seja incluida no cabeçalho de cada arquivo. O conteúdo de globais.h pode ser visto abaixo:

```
#ifndef SOURCE_CODES_GLOBAIS_H_

    #define SOURCE_CODES_GLOBAIS_H_

    #include "DSP28x_Project.h" // Device Headerfile and Examples
Include File
```

```
#include "f2802x_common/include/clk.h"
#include "f2802x_common/include/flash.h"
#include "f2802x_common/include/gpio.h"
#include "f2802x_common/include/pie.h"
#include "f2802x_common/include/pll.h"
#include "f2802x_common/include/pwm.h"
#include "f2802x_common/include/wdog.h"
#include "f2802x_common/include/spi.h"
#include "f2802x_common/include/adc.h"
#include "iir.h"
#include <IQmathLib.h>
#define GLOBAL_Q 20 // seleciona a base 20 como GLobal Q

extern CLK_Handle myClk;
extern FLASH_Handle myFlash;
extern GPIO_Handle myGpio;
extern PIE_Handle myPie;
extern PWM_Handle myPwm1;
extern PWM_Handle myPwm2;
extern ADC_Handle myAdc;
extern CPU_Handle myCpu;
extern SPI_Handle mySpi;
extern PLL_Handle myPll;
extern WDOG_Handle myWDog;

#endif /* SOURCE_CODES_GLOBAIS_H_ */
```

5.2 Programa 5

O programa 5 consiste no trabalho do aluno André Cunha. O programa era demasiadamente grande por isso o uso da modularização nele se enquadrou perfeitamente. Assim foi possível separar as partes do código e melhorar a organização. Como foram necessário quatro arquivos contendo várias linhas de código optou-se por criar um arquivo zip do projeto e o link para download é:
https://mega.nz/#!9FwAHYKA!
hg1QBMVEUd4Ib7VFzLTQNvFzSClPYqunvBW2ZGBmwAw

ANEXO 1

```
#include "DSP28x_Project.h"
#include "f2802x_common/include/clk.h"
#include "f2802x_common/include/flash.h"
#include "f2802x_common/include/gpio.h"
#include "f2802x_common/include/pie.h"
#include "f2802x_common/include/pll.h"
#include "f2802x_common/include/wdog.h"


void main(void)
{
   //Declara os Handles que serão usados na função Main
   CLK_Handle myClk;
   CPU_Handle myCpu;
   FLASH_Handle myFlash;
   GPIO_Handle myGpio;
   PIE_Handle myPie;
   PLL_Handle myPll;
   WDOG_Handle myWDog;

   // Inicializa os Handles necessarios para esta aplicação
   myClk = CLK_init((void *)CLK_BASE_ADDR, sizeof(CLK_Obj));
   myCpu = CPU_init((void *)NULL, sizeof(CPU_Obj));
   myFlash = FLASH_init((void *)FLASH_BASE_ADDR, sizeof(FLASH_Obj));
   myGpio = GPIO_init((void *)GPIO_BASE_ADDR, sizeof(GPIO_Obj)); myPie
   = PIE_init((void *)PIE_BASE_ADDR, sizeof(PIE_Obj)); myPll = PLL_init((void
   *)PLL_BASE_ADDR, sizeof(PLL_Obj));
   myWDog = WDOG_init((void *)WDOG_BASE_ADDR, sizeof(WDOG_Obj));

   /  Executa a inicialização basica
   WDOG_disable(myWDog);
   CLK_enableAdcClock(myClk);
   (*Device_cal)();

   //Seleciona o Oscilador externo como fonte de Clock
   CLK_enableCrystalOsc (myClk);
   PLL_enableOsc (myPll);
   CLK_disableClkIn(myClk);
   CLK_enableCrystalOsc(myClk);
   CLK_setOsc2Src(myClk, CLK_Osc2Src_External);
   CLK_setOscSrc(myClk, CLK_OscSrc_External);
```

```
   / Coloca o multiplicador da PLL por 10(dez) e o divisor por 2(dois) que gera
(12MHz*10)/2 = 60MHz
   PLL_setup(myPll, PLL_Multiplier_10, PLL_DivideSelect_ClkIn_by_2);

   / Desabilita a PIE e todas as interrupções
   PIE_disable(myPie);
   PIE_disableAllInts(myPie);
   CPU_disableGlobalInts(myCpu);
   CPU_clearIntFlags(myCpu);

   //Configura o GPIO 18 no modo XCLKOUT que mostra o clock
   GPIO_setPullUp(myGpio, GPIO_Number_18, GPIO_PullUp_Enable);
   GPIO_setQualification(myGpio, GPIO_Number_18, GPIO_Qual_ASync);
   GPIO_setMode(myGpio, GPIO_Number_18, GPIO_18_Mode_XCLKOUT);

CLK_setClkOutPreScaler(myClk,CLK_ClkOutPreScaler_SysClkOut_by_4);
//Divide o clock por 4, assim no GPIO18 a frequência deve ser 15MHz

   / Inicializa a memoria
Flash #ifdef _FLASH
   memcpy(&RamfuncsRunStart,
&RamfuncsLoadStart, (size_t)&RamfuncsLoadSize);
   FLASH_setup(myFlash);
#endif

   while(1)
        {
        //loop infinito
        }
}
```

ANEXO 2

```
#include "DSP28x_Project.h"
#include "f2802x_common/include/clk.h"
#include "f2802x_common/include/flash.h"
#include "f2802x_common/include/gpio.h"
#include "f2802x_common/include/pie.h"
#include "f2802x_common/include/pll.h"
#include "f2802x_common/include/spi.h"
#include "f2802x_common/include/wdog.h"


/  Declaração dos protótipos de funções encontradas no
arquivo void Calc_filt(void);
void delay_loop(void);
void spi_xmit(uint16_t a);
void spi_fifo_init(void);
void spi_init(void);
void error(void);
void CLK_setLowSpdPreScaler2(CLK_Handle clkHandle,
const CLK_LowSpdPreScaler_e preScaler);


void main(void)
{
   uint16_t sdata; // Dado envisado(sent)
   uint16_t rdata; // Dado recebido(received)
   /  Caso seja usado interrupções para o uso do SPI e seja necessario
verificar os dados recebidos e enviados
   /  as variaveis devem ser globais para serem usadas nas funções
de interrupção e não terem seus valores perdidos

   //Declara os Handles que serão usados na função Main
   CLK_Handle myClk;
   CPU_Handle myCpu;
   FLASH_Handle myFlash;
   GPIO_Handle myGpio;
   PIE_Handle myPie;
   PLL_Handle myPll;
   SPI_Handle mySpi;
   WDOG_Handle myWDog;

   // Inicializa os Handles necessarios para esta aplicação
   myClk = CLK_init((void *)CLK_BASE_ADDR, sizeof(CLK_Obj));
   myCpu = CPU_init((void *)NULL, sizeof(CPU_Obj));
```

```
myFlash = FLASH_init((void *)FLASH_BASE_ADDR, sizeof(FLASH_Obj));
myGpio = GPIO_init((void *)GPIO_BASE_ADDR, sizeof(GPIO_Obj)); myPie
= PIE_init((void *)PIE_BASE_ADDR, sizeof(PIE_Obj)); myPll = PLL_init((void
*)PLL_BASE_ADDR, sizeof(PLL_Obj));
mySpi = SPI_init((void *)SPIA_BASE_ADDR, sizeof(SPI_Obj));
myWDog = WDOG_init((void *)WDOG_BASE_ADDR, sizeof(WDOG_Obj));

/  Executa a inicialização basica
WDOG_disable(myWDog);
CLK_enableAdcClock(myClk);
(*Device_cal)();

CLK_enableCrystalOsc (myClk);
PLL_enableOsc (myPll);
CLK_disableClkIn(myClk);
CLK_enableCrystalOsc(myClk);
CLK_setOsc2Src(myClk, CLK_Osc2Src_External);

//Seleciona o Oscilador interno 1 como fonte de Clock
CLK_setOscSrc(myClk, CLK_OscSrc_Internal);

/  Coloca o multiplicador da PLL por 6(seis) e o divisor por 1(um) que gera
(10MHz*6)/1 = 60MHz
PLL_setup(myPll, PLL_Multiplier_10, PLL_DivideSelect_ClkIn_by_2);
/  Desabilita a PIE e todas as interrupções
PIE_disable(myPie);
PIE_disableAllInts(myPie);
CPU_disableGlobalInts(myCpu);
CPU_clearIntFlags(myCpu);

/  Caso esteja rodando em flash copia as funções exclusivas da RAM para a
RAM
#ifdef _FLASH
memcpy(&RamfuncsRunStart,
&RamfuncsLoadStart, (size_t)&RamfuncsLoadSize);
FLASH_setup(myFlash);
#endif


/  Inicializa a GPIO
    //Escolhe o PullUp de cada pino
GPIO_setPullUp(myGpio,   GPIO_Number_16,   GPIO_PullUp_Enable);
GPIO_setPullUp(myGpio,   GPIO_Number_17,   GPIO_PullUp_Enable);
GPIO_setPullUp(myGpio, GPIO_Number_18, GPIO_PullUp_Enable);
```

```
    GPIO_setPullUp(myGpio, GPIO_Number_19, GPIO_PullUp_Enable);

        //Escolhe
    GPIO_setQualification(myGpio,   GPIO_Number_16,   GPIO_Qual_ASync);
    GPIO_setQualification(myGpio,   GPIO_Number_17,   GPIO_Qual_ASync);
    GPIO_setQualification(myGpio,   GPIO_Number_18,   GPIO_Qual_ASync);
    GPIO_setQualification(myGpio, GPIO_Number_19, GPIO_Qual_ASync);

        //Escolhe o modo de funcionamento de cada pino
    GPIO_setMode(myGpio, GPIO_Number_16, GPIO_16_Mode_SPISIMOA);
    GPIO_setMode(myGpio, GPIO_Number_17, GPIO_17_Mode_SPISOMIA);
    GPIO_setMode(myGpio, GPIO_Number_18, GPIO_18_Mode_XCLKOUT);
    GPIO_setMode(myGpio, GPIO_Number_19,
GPIO_19_Mode_SPISTEA_NOT);

    PIE_setDebugIntVectorTable(myPie);
    PIE_enable(myPie);
                        // Inicializa o SPI
    spi_init();          // Inicializa as FIFOs do SPI
    spi_fifo_init();


    sdata = 0x0000;

    for(;;) {

        // Manda informação
        SPI_write(mySpi, sdata); // Escreve sdata no registro
SPITXBUF(Serial Transmit Buffer Register)

        // Espera até que o dado tenha sido recebido
        while(SPI_getRxFifoStatus(mySpi) == SPI_FifoStatus_Empty)
        {
        }

        /  Checa a informação recebida
        rdata = SPI_read(mySpi);
        if(rdata != sdata)
          {
           error();        //Erro no loop back
          }
        sdata++;
    }
}
```

```
void delay_loop()
{
    long      i;

    for (i = 0; i < 1000000; i++) {
    }

    return;
}

void error(void)
{
    asm(" ESTOP0");     // Falha no teste! Pare!
    for (;;){
    }
}

void spi_init()
{
    CLK_enableSpiaClock(myClk);

    // Reset on, rising edge, 16-bit char bits
    SPI_setCharLength(mySpi, SPI_CharLength_16_Bits); // Reinicia o
envio quando, borda de subida, Char de 16 bits for enviado

    /  Enable master mode, normal phase,
    /  enable talk, and SPI int disabled.
    /  Habilita o modo mestre, fase normal
    SPI_setMode(mySpi,
    SPI_Mode_Master); //Habilita transmitir
    SPI_enableTx(mySpi);

    /  Configura o divisor do Clk para formar o LSPCLK(Low Speed
Peripheral Clock)

CLK_setLowSpdPreScaler2(myClk,CLK_LowSpdPreScaler_SysClkOut_by_1);

          /  O valor do clock do SPI é controlado pelo SPI Baud Rate
  Register(SPIBRR) que pode ser visto em SPRUG71B - Pag 28/39
    SPI_setBaudRate(mySpi, SPI_BaudRate_1_MBaud); // Seta o valor do
SPIBRR
    /  É importante criar no arquivo spi.h os valores de SPIBRR desejados
para conseguir obter a frequencia desejada
```

```
    /  Criar em SPI_Priority_e que fica dentro de SPI.h. O valor do clock do SPI
é igual a LSPCLK/(SPIBRR+1) para SPIBRR entre 3 e 127
    /  Para SPIBRR= 0;1;2 o clock do SPI será LSPCLK/4

    /  Habilita o Loopback
    SPI_enableLoopBack(mySpi);

    //Liga o SPI
    SPI_enable(mySpi);

    /  Coloca de forma que breakpoints de emulação não interrompam
a transmissão
    SPI_setPriority(mySpi, SPI_Priority_FreeRun);

    return;
}

void spi_fifo_init()
{

    /  Initialize SPI FIFO registers
    /  Inicializa os registradores FIFO do SPI
    SPI_enableChannels(mySpi);
    SPI_enableFifoEnh(mySpi); SPI_resetTxFifo(mySpi);
    SPI_clearTxFifoInt(mySpi); SPI_resetRxFifo(mySpi);
    SPI_clearRxFifoInt(mySpi);
    SPI_setRxFifoIntLevel(mySpi,
    SPI_FifoLevel_4_Words);


    return;
}
void CLK_setLowSpdPreScaler2(CLK_Handle clkHandle,
const CLK_LowSpdPreScaler_e preScaler) {

    CLK_Obj *clk = (CLK_Obj *)clkHandle;


    ENABLE_PROTECTED_REGISTER_WRITE_MODE; //Habilita escrita em
registro trotegido

    // set the bits
```

```
    clk->LOSPCP &= 0XFFF8; // Apenas os bits de LSPCLK foram
zerados(Bits 3-15 de LOSPCP são reservados, eles não gostam de socializar
kkkkkk)
    clk->LOSPCP |= preScaler; //Só a parte do LSPCLK de LOSPCP foi
alterada
    //O regisro LOSPCP pode ser visto de melhor forma em
SPRUFN3D-Pag:35/139

    DISABLE_PROTECTED_REGISTER_WRITE_MODE; //Habilita leitura em
registro trotegido

    return;
}
```

ANEXO 3

```
#include "DSP28x_Project.h"
#include <math.h> //Usada apenas para gerar o sinal senoidal que sera visto
mais a frente
#include "f2802x_common/include/clk.h"
#include "f2802x_common/include/flash.h"
#include "f2802x_common/include/pie.h"
#include "f2802x_common/include/pll.h"
#include "f2802x_common/include/wdog.h"
#include "iir.h"

#define SIGNAL_LENGTH 100 // Define quantas posições tera o vetor de
amostragem
int32_t sigIn[SIGNAL_LENGTH];

IIR5BIQ32 iir = IIR5BIQ32_DEFAULTS; // Cria uma variavel iir do tipo
IIR5BIQ32 e armazena nela IIR5BIQ32_DEFAULTS

//Configura o tamanho do COEFFICIENT BUFFER e o DELAY BUFFER const
   int32_t coeff[5*IIR32_LPF_NBIQ] = IIR32_LPF_COEFF; // O tamanho
do vetor de coeficientes é igual a 5 * IIR32_LPF_NBIQ
   int32_t dbuffer[2*IIR32_LPF_NBIQ];


void main(void)
{

   //Declara os Handles que serão usados na função Main
   CLK_Handle myClk;
   CPU_Handle myCpu;
   FLASH_Handle myFlash;
   PIE_Handle myPie;
   PLL_Handle myPll;
   WDOG_Handle myWDog;

   // Inicializa os Handles necessarios para esta aplicação
   myClk = CLK_init((void *)CLK_BASE_ADDR, sizeof(CLK_Obj));
   myCpu = CPU_init((void *)NULL, sizeof(CPU_Obj));
   myFlash = FLASH_init((void *)FLASH_BASE_ADDR, sizeof(FLASH_Obj));
   myPie = PIE_init((void *)PIE_BASE_ADDR, sizeof(PIE_Obj)); myPll =
   PLL_init((void *)PLL_BASE_ADDR, sizeof(PLL_Obj));
   myWDog = WDOG_init((void *)WDOG_BASE_ADDR, sizeof(WDOG_Obj));

   // Executa a inicialização basica
```

```
    WDOG_disable(myWDog);
    CLK_enableAdcClock(myClk);
    (*Device_cal)();

    //Seleciona o Oscilador interno 1 como fonte de Clock
    CLK_setOscSrc(myClk, CLK_OscSrc_Internal);

    /  Coloca o multiplicador da PLL por 6(seis) e o divisor por 1(um) que gera
(10MHz*6)/1 = 60MHz
    PLL_setup(myPll, PLL_Multiplier_6, PLL_DivideSelect_ClkIn_by_1);

    /  Desabilita a PIE e todas as interrupções
    PIE_disable(myPie);
    PIE_disableAllInts(myPie);
    CPU_disableGlobalInts(myCpu);
    CPU_clearIntFlags(myCpu);

    /  Caso esteja rodando em flash copia as funções exclusivas da RAM para a
RAM
#ifdef _FLASH
    memcpy(&RamfuncsRunStart,
&RamfuncsLoadStart, (size_t)&RamfuncsLoadSize);
    FLASH_setup(myFlash);
#endif

    /  Inicializa o tabela de vetores do debug e habilita a PIE
    PIE_setDebugIntVectorTable(myPie);
    PIE_enable(myPie);

    /  Paramatros da Senoide
        int32_t xn = 0, yn = 0;
        float TimeStep = 0.00016666666666666667f; // Seu time step, caso
esteja simulando deve ser o inverso de sua frequência de amostragem
        float Time = 0.0f; // Tempo inicial. É possivel modifica-lo para criar
uma senoide defasada

        /  Inicialização do Filtro IIR
        iir.dbuffer_ptr = dbuffer;
        iir.coeff_ptr = (long *)coeff;
        iir.qfmat       = IIR32_LPF_QFMAT;
        iir.nbiq        = IIR32_LPF_NBIQ;
        iir.isf         = IIR32_LPF_ISF;
        iir.init(&iir);
```

```
        //Simulando uma senoide de 60Hz como sinal como a frequência
de amostragem sendo 6KHz
        uint32_t i = 0; //Variável auxiliar
        for(i=0; i < SIGNAL_LENGTH; i++)
        {
          xn = (long)(2147483648*(sin(376.99111843077515*Time))); // A
entrada do filtro deve ser normalizada em Q31 e 2147483648 é 1 em Q31
          sigIn[i]   = xn;
          Time       = Time + TimeStep;
        }

    for(;;) //Loop infinito na main
    {
                    i++
                    if(i>=SIGNAL_LENGTH)
                    {
                            i=0;
                    }
                /  Pega a senoide gerada anteriormente e joga na entrada do filtro
e calcula a saída
                    iir.input = sigIn[i];
                    iir.calc(&iir);
                    yn = iir.output32; // É importante notar que a saida do filtro é
dada em Q30 para output32 e Q14 para output16
                    //Ou seja caso de interesse comparar as variaveis yn deve
ser multiplicado por dois
    }
}
```

ANEXO 4

```
#include "DSP28x_Project.h"
#include "f2802x_common/include/clk.h"
#include "f2802x_common/include/flash.h"
#include "f2802x_common/include/pie.h"
#include "f2802x_common/include/pll.h"
#include "f2802x_common/include/wdog.h"

/  Declaração dos protótipos de funções encontradas no
arquivo void DelayRAM(void);
void DelayFLASH(void);

/  Declara quais funções serão transferidas para a memoria RAM ao iniciar o
programa
#pragma CODE_SECTION(DelayRAM, "ramfuncs");

void main(void)
{

    //Declara os Handles que serão usados na função Main

    CLK_Handle myClk;
    CPU_Handle myCpu;
    FLASH_Handle myFlash;
    PIE_Handle myPie;
    PLL_Handle myPll;
    WDOG_Handle myWDog;

    // Inicializa os Handles necessarios para esta aplicação
    myClk = CLK_init((void *)CLK_BASE_ADDR, sizeof(CLK_Obj));
    myCpu = CPU_init((void *)NULL, sizeof(CPU_Obj));
    myFlash = FLASH_init((void *)FLASH_BASE_ADDR, sizeof(FLASH_Obj));
    myPie = PIE_init((void *)PIE_BASE_ADDR, sizeof(PIE_Obj)); myPll =
    PLL_init((void *)PLL_BASE_ADDR, sizeof(PLL_Obj));
    myWDog = WDOG_init((void *)WDOG_BASE_ADDR, sizeof(WDOG_Obj));

    /  Executa a inicialização basica
    WDOG_disable(myWDog);
    CLK_enableAdcClock(myClk);
    (*Device_cal)();

    //Seleciona o Oscilador interno 1 como fonte de Clock
    CLK_setOscSrc(myClk, CLK_OscSrc_Internal);
```

```
   / Coloca o multiplicador da PLL por 6(seis) e o divisor por 1(um) que gera
(10MHz*6)/1 = 60MHz
   PLL_setup(myPll, PLL_Multiplier_6, PLL_DivideSelect_ClkIn_by_1);

   / Desabilita a PIE e todas as interrupções
   PIE_disable(myPie);
   PIE_disableAllInts(myPie);
   CPU_disableGlobalInts(myCpu);
   CPU_clearIntFlags(myCpu);

   / Inicializa a memoria Flash e transfere as funções da RAM para a mémoria
RAM
#ifdef _FLASH
   memcpy(&RamfuncsRunStart,
&RamfuncsLoadStart, (size_t)&RamfuncsLoadSize);
   FLASH_setup(myFlash);
#endif


   / Inicializa o tabela de vetores do debug e habilita a PIE
   PIE_setDebugIntVectorTable(myPie);
   PIE_enable(myPie);

   for(;;)
   {
         DelayRAM();
         DelayFLASH();
   }
}

void DelayRAM(void)
{
         int i;
         for(i=0;i<1000;i++)
         {
         }
}

void DelayFLASH(void)
{
         int i;
         for(i=0;i<1000;i++)
         {
         }
```

}